A Tactical and Strategic AI Interface for Real-Time Strategy Games

Michael van Lent, Paul Carpenter, Ryan McAlinden, Poey Guan Tan

University of Southern California
Institute for Creative Technologies
13274 Fiji Way
Marina del Rey, CA 90292
[vanlent, carpenter, mcalinden]@ict.usc.edu

DSO National Laboratories
20 Science Park Drive
Singapore 118230
tpoeygua@dso.org.sg

Abstract

Real Time Strategy (RTS) games present a wide range of AI challenges at the tactical and strategic level. Unfortunately, the lack of flexible "mod" interfaces to these games has made it difficult for AI researchers to explore these challenges in the context of RTS games. We are addressing this by building two AI interfaces into Full Spectrum Command, a real time strategy training aid built for the U.S. Army. The tactical AI interface will allow AI systems, such as Soar and Simulation Based Tactics Mining, to control the tactical behavior of platoons and squads within the environment. The strategic AI interface will allow AI planners to generate and adapt higher-level battle plans which can in turn be executed by the tactical AI. This paper describes these two interfaces and our plans for identifying and addressing the research challenges involved in developing and deploying tactical and strategic AI systems.

Introduction

The majority of game-related artificial intelligence research projects have focused on First Person Shooter (FPS) games such as Quake II (Laird 2001), Half-Life (Khoo et al. 2002), and Unreal Tournament (Young et al 2004). To a large extent this is due to the flexibility and power of the publicly available "mod" interfaces built into games in this genre. These mod interfaces allow researchers to quickly interface their AI architectures with the game engine and tailor the game environment to meet their needs. Unfortunately, it turns out that reaction time and aiming skill are some of the most important factors in success in an FPS game. Deep reasoning about tactics and strategy do not end up playing as big a role as might be expected. This limits the relevant AI challenges to considerations such as real-time reactivity, aiming, and steering behaviors without considering some of the higher-level activities such as group coordination and mission intent.

In order to move towards deeper AI challenges, a number of projects have used FPS game engines to build completely different types of environments that incorporate a greater range of AI capabilities. This is a creative and effective solution that allows the researchers to customize the game to their research interests and demonstrate how novel approaches to AI can support new game features and genres. To date the majority of these game mods for research have focused on interpersonal social interaction (Laird et al. 2002, Gordon et al. 2004) and interactive narrative (Young et al. 2004).

However, when the focus of the AI research effort is tactical behavior and strategic reasoning a different game genre, Real-Time Strategy (RTS), represents a more suitable environment. RTS games involve both strategic reasoning, where the AI controller is developing a high-level battle plan, and tactical behavior, when the AI units are executing the orders contained in the battle plan. Unfortunately, the mod interfaces in RTS games are not as flexible as FPS mod interfaces particularly when it comes to interfacing external AI architectures. This is primarily a result of RTS games that cover a broad range of missions and goals. For example, FPS games typically include a player and NPC units whose sole mission is to destroy others around them. Weapon types, terrain, and arena (i.e. futuristic, historical conflicts) are independent of the AI. However, RTS games must take these additional aspects into account for each game, thus catering the AI to a specific environment.

This paper describes an ongoing effort at the University of Southern California's Institute for Creative Technologies (ICT) to build both a tactical AI interface

and a strategic AI interface into the military-themed Real Time Strategy game, Full Spectrum Command (FSC). FSC is a "commercial platform training aid" developed by ICT and Quicksilver software for the U.S. Army. Unlike a game, which is designed purely for entertainment, FSC is a training aid designed to help teach officers the cognitive and leadership skills required to command a light infantry company. While entertainment is not the primary goal, it is a useful tool in achieving the primary training objectives despite looking and playing much like a real time strategy game seen on the market today.

In FSC the player/student takes on the role of a U.S. Army Captain commanding a light infantry company of about 120 soldiers. A mission in FSC consists of three phases of game play. The first, called the planning phase, requires a great deal of strategic reasoning to generate a battle plan. In the current version of FSC this strategic reasoning is all done by the human player (for the Army side) or the human mission designer (for the enemy side). The second phase, called the execution phase, requires a great deal of tactical behavior as the battle plan is executed, as well as some strategic reasoning to adapt the battle plan on the fly. The third phase, the After Action Review (AAR), allows the instructor and the player to examine what happened during the course of the mission, and to analyze the decisions made. In the current version of FSC all of the tactical behavior is done by an ad-hoc AI system created by the game developers at Quicksilver software (van Lent et al. 2004). The strategic adaptation is done by the human player (for the Army side) while the enemy currently has no ability to adapt.

Our goal in this work is to build a general tactical AI interface and a general strategic AI interface into FSC that will allow researchers to experiment with and compare different AI systems in these two roles. With the tactical AI interface, researchers will be able to replace the current "game AI" system with a range of AI architectures such as Soar (Lehman) and Simulation Based Tactics Mining (Sui 2000). Likewise, with the strategic AI interface, researchers will be able to explore the use of AI planners, such as SHOP2 (Nau et al. 2003) and VHPOP (Hakan et al. 2003), to replace humans in the strategic reasoning and adaptation role.

Comparing Tactical AI Systems

At the tactical behavior level we will compare two systems, the Soar architecture and Simulation Based Tactics Mining. This section describes each of these systems as well as the common tactical AI interface they will both use to communicate with the FSC game engine.

Tactical AI Systems

Company-level tactical behavior in FSC focuses on the activities of platoons, which are further broken down into squads, fire-teams and individual soldiers. Our tactical AI interface allows external systems to control both platoon (about 34 soldiers) and squad-level (9 soldiers) behaviors.[1] The tactical AI system receives events from the game that trigger a decision cycle (NPCSpotted, FiredAt). At the start of each cycle, game state information is then retrieved to populate preconditions (NPC Locations, Ammo status). The decision cycle then runs and outputs (at the highest level) platoon-level orders such as "Move along a route", "Breach an obstacle", "Clear a building", "Support another unit with covering fire", "Deploy smoke" and "Assault and destroy the enemy in a specified region." From here, the AI system then decomposes these fairly general orders into specific commands for each squad (and in turn fire team). It can take a multi-agent approach (one agent per platoon and squad) or a single agent approach (one monolithic agent controlling all platoons and squads). Fire-team tasks include "Move to a point", "Fire at target", "Fire at area" and "Take cover" among others, which are handed off to the game's AI responsible for actual path-planning, steering, and animation control.

As part of this comparison, two tactical AI systems are being examined: Soar is an integrated architecture for knowledge-based problem solving, learning, and interaction with external environments; Simulation Based Tactics Mining (SBTM), developed at the DSO National Laboratories (Singapore), is an adaptive learning, rule-based decision making system.

Soar has been used to drive intelligent AI in a variety of virtual environments, such as ModSAF, Quake II and Unreal Tournament. It takes a rule-based approach encoding behaviors as operators each consisting of one or more proposal rules (specifying preconditions) and one or more application rules (specifying effects). These rules interact with short and long-term memory to generate behavior commands. Long-term memory contains Soar's rules, called productions, while short-term memory contains the sensor input, overall situation awareness, mission, and intermediate objectives. Intelligent actions require in-depth information about the environment, which comes through an input-link (part of short-term memory) responsible for extracting sensor data from the game. More information on these sensors is presented below.

Each cycle through Soar's "sense, think, act" loop, or decision cycle, processes the sensor information, matches the rules to fire operators which send actions back across the output link to the game for execution. In most previous Soar systems the actions control an individual Non-Player Character (NPC). This is in contrast to fire team-level actions described in the next section, which will be issued by Soar and decomposed into individual NPC actions within FSC.

Simulation Based Tactics Mining (SBTM) focuses on automatically generating new rules from existing rule databases. It learns by interacting with the environment using machine learning techniques such as genetic

[1] Because the Fire Team (4-5 soldiers) and individual soldier behaviors are so closely tied to the game engine the pre-existing FSC AI will handle these.

algorithms and reinforcement learning. Reinforcement learning is used in the evaluation of rules, while genetic algorithms provide an effective means of generating new rules and exploring the large search space. By complementing the search and generation capabilities of genetic algorithms and the unsupervised nature of reinforcement learning, SBTM is able to direct the evolution of rules towards optimality.

SBTM is built upon the Advanced Rule Engine (ARE), a rule-based decision making system. Developed at DSO National Laboratories (Singapore), ARE has the ability to handle uncertain, incomplete and fuzzy inputs. In the domain of computer games, ARE has been developed as an interface to the Half Life FPS game using the "Flexbot mod." The sensor inputs ARE extracts from the game are similar to Soar's, which are described in the next section.

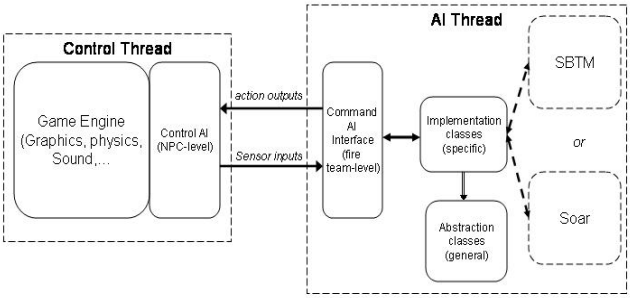
The individual actions of a Fire Team are determined by ARE's decision cycle, such as moving to a specific location and firing at an enemy in a certain direction, which is based on the rules written in conjunction with input from the sensor information. The performance of ARE and any other rule-based engine is highly dependent on the quality of the rules. The knowledge engineering process required to create high-quality rules is tedious and time consuming. SBTM addresses this problem by using the genetic algorithms and reinforcement learning to automate the rule generation process. A full description of ARE and SBTM are outside of the scope of this paper and the reader is referred to Sui et al (Sui et al. 2000) for more details.

In general, SBTM is applicable in a variety of domains and has been specifically applied and evaluated in air combat simulations. SBTM has also been shown to evolve rules that, in some instances, exhibit performance superior to hand crafted rules solicited from subject matter experts.

Tactical AI Interface

The interface that has been designed to support the integration of tactical AI systems, such as Soar and the SBTM, is twofold. First, it intends to strip away any architecture-specific details regarding mission representation or task execution. For example, Soar can take in certain baseline sensor information about the environment (NPC locations, line of sight calculations, weapon types) and build an internal hierarchical working memory structure (a group of NPCs organized as a team with an overall objective). This hierarchical representation may not be suitable for something like the SBTM, and so these baseline sensor inputs must be abstracted away by either of the two systems, which internally construct the representation needed. For example, SBTM requires additional feedback information for evaluation during the adaptive learning and the rule generation process. This information includes time taken to accomplish a mission and the number of enemy and friendly casualties at the end of the mission.

This abstraction consists of a series of interface classes aimed to manage various aspects of the game's AI. As with any game AI, it is necessary to input and represent the game's data model, such as character locations and mission status, as well as the terrain. These interface classes are split into two layers, an abstraction layer and an implementation layer, making it generic enough to be piped to other AI systems. The abstraction layer defines the framework that the game engine and the AI system adhere to. It defines AI interfacing functions, sensor extraction functions, and action actuation functions. The implementation layer is game engine and AI system-specific and is extended from the abstraction classes. These implementation classes process game information to the AI system (sensor inputs), and interpret the actions made in the AI decision cycle (action outputs), which are sent to the environment for execution.



The tactical AI for this evaluation will cover the variety of traditional platoon and squad level tasks described earlier. However, Soar and the SBTM will not control NPCs at the individual level. Instead, actions and tasks determined within each decision cycle will be sent to the game engine at the fire-team level. It will then be up to the existing control AI within the game to decompose these unit-level actions into individual NPC actions (FireAt, MoveTo) for execution.

This involves some challenging issues for the tactical AI such as teamwork and coordination between fire teams, squads and platoons. Sensor inputs cannot simply be represented for individual NPC's, but instead must be combined with other NPC sensor data to draw a group conclusion about the state of the world. For example, a friendly NPC casualty within a fire-team is of higher concern to the NPC's fire team, but not to the overall platoon.

Specifically, Soar will use the input/output mechanisms described in the previous section to manage incoming data and outgoing actions within the game. The interface will take in (on the input link) sensor value pairs required by Soar in its decision cycle:

- Friendly Team composition
- Current Mission (and status)
- Equipment
- Location

As discussed above, upon completion of its decision cycle, fire-team level actions will then be sent back to the engine through the interface for control AI decomposition and execution. These actions include:

- Move To (route)
- Fire at Enemy
- Face Target
- Clear Room (requires decomposition)

The implementation of the interface is ongoing, though it is being designed to use an event-driven mechanism for the AI's decision cycle. The AI System (Soar, SBTM) will run in a separate thread that is tied to the interface; each time an event of interest is passed to the interface (NPCSpotted), the AI system queries the game for additional precondition information (NPC Locations), which then populates the input-link. Upon completion of the decision cycle, the output is immediately returned to the Interface Thread and to the game for execution.

Despite the differences in SOAR's and SBTM's architecture, the sensor inputs and action outputs that are required by the two engines are fairly similar: both engines operate at the fire-team level, and both require information not only about characters in the environment, but also overall mission goals and objectives. The abstraction layer and implementation layer described here will isolate any differences and will promote the use of other AI systems in the future.

Strategic AI Systems

The Strategic AI System generates plans to actualize high-level mission goals. Plan operators correspond to the platoon-level tasks that are executed by the tactical AI. Up-to-date world state information received from the game engine is used to assign values to operator predicates. To support the widest range of planners available, the domain is defined using the language specified for use in the International Planning Competition – PDDL 2.1 (Fox & Long 2003).

The three major challenges we face when developing the Strategic AI System are terrain analysis, uncertain operator durations, and multi-agent coordination of plans. The terrain must be taken into account when choosing a plan of attack – certain paths may provide better cover while other avenues of attack may be more direct. Varying avenues of attack obviously cause uncertainty in the duration of the operators. Other environmental features unknown at the planning stage also cause uncertainties – only monitoring can determine when the operators complete execution. Monitoring is also required in the coordination of the team plans. Often times, one team cannot begin executing an action prior to the completion of a second team's actions.

Mission objectives are divided into two categories – terrain-based missions and force-based missions. Terrain-based goals include assault, defend, recon, and secure missions. Force-based goals are different because they do not focus on specific points within the terrain; instead, they focus on destroying OPFOR units, surviving an OPFOR attack, or rescuing/escorting civilians. Once a mission objective is chosen by the user, it is communicated to the planning system along with any initial world state information.

FSC includes utilities for humans to generate mission plans. A plan, called an execution matrix, consists of sequences of platoon level tasks ordered by phases. Examples of actions include moving along a route, clearing a building, performing support by fire, etc… These are the same tasks that can be assigned to the tactical AI. At the beginning of a mission the strategic AI will generate a plan (sequence of tasks for each platoon) that will achieve the mission objective from the initial world state. The generation of the plan may be additionally constrained by rules of engagement imposed on the strategic AI. The plan is then written to disk as an execution matrix. The format of the execution matrix defines the interface between the external Strategic AI System and the game engine. Instead of a human defining plans by hand, the plans are generated by the Strategic AI System based on the mission objective and the initial world state.

However, the strategic AI's job does not end once the initial execution matrix is generated. The strategic AI must monitor the execution of the plan to coordinate the actions of the teams. As such, operator predicates are set using world state information from the game engine. These predicates are used to determine when execution of a platoon's task has finished. Coordination is handled by the strategic AI issuing "go-codes" that are understood by the game engine. In the event of a plan failure, a new plan can be generated using the current world state information and communicated to the game engine replacing the previous execution matrix with the newly generated one.

Future Work

At this point the development of the tactical AI and strategic AI systems are very much "works in progress." The interface designs for both systems are complete and the modifications to Full Spectrum Command are well underway. The initial tactical AI systems will be built on the existing foundations of the Soar architecture and the SBTM. The initial strategic AI systems will be extensions of well established planning algorithms. As with many research projects the road ahead looks clear but hurdles will inevitably pop up and provide interesting research challenges for years to come.

References

Fox, M. and Long, D. 2003 "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains", *Journal of Artificial Intelligence Research*, Volume 20, pages 61-124.

Gordon, A., van Lent, m., van Velsen, M., Carpenter, P., and Jhala, A. 2004. "Branching Storylines in Virtual Reality Environments for Leadership Development", *The*

Sixteenth Innovative Applications of Artificial Intelligence Conference, August 2004.

Håkan L. S. Younes and Reid G. Simmons. 2003. VHPOP: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research*, 20:405-430, 2003.

Khoo, A. and Zubek, R., 2002. "Applying Inexpensive AI Techniques to Computer Games" *IEEE Intelligent Systems*, 2002, 17(4).

Lehman, J.F., Laird, J.E., & Rosenbloom, P.S. (In press) A gentle introduction to Soar, an architecture for human cognition. In S. Sternberg & D. Scarborough (eds.) *Invitation to Cognitive Science, Volume 4.*

Laird, J., 2001. "It knows what you are going to do: Adding anticipation to a Quakebot." *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, Canada, May 28-June 1, 2001.

Laird, J. E., et al. 2002. "A Test Bed for Developing Intelligent Synthetic Characters", AAAI Spring Symposium on AI and Interactive Entertainment, March 25-27, 2002.

Nau, D.S., Au, T.C., Ilghami, O., Kuter, U., Murdock, J.W., Wu, D. and Yaman, F. 2003. "SHOP2: An HTN Planning System", *Journal of Artificial Intelligence Research*, Volume 20, pages 379-404.

Sui Q., How K.Y., and Ong W.S., 2000. "An intelligent agent in an air combat domain", *4th International Conference on Autonomous Agent*, January 1, 2000.

van Lent, m., Fisher, B. and Mancuso, M., 2004. "An Explainable Artificial Intelligence System for Small-unit Tactical Behavior", *The Sixteenth Innovative Applications of Artificial Intelligence Conference*, August 2004.

Young, R. M., Riedl, M. O., Branly, M., Jhala, A., Martin, and R. J., Saretto, C. J., 2004. "An architecture for integrating plan-based behavior generation with interactive game environments", *Journal of Game Development*, 2004, 1(1).